2

Technical Report ICMA-91-157 March 1991

NFEARS
A Nonlinear Adaptive Finite Element Solver[1]

Part II: User's Manual
(Version 6)

by

Charles K. Mesztenyi[2] and Werner C. Rheinboldt[3]

[1] This work was in part supported by the Office of Naval Research under contracts N-00014-80-C-9455 and N-00014-85-K-0169 and by the National Science Foundation under grant DCR-8309926. Acknowledgement is also made for the partial support of the Computer Science Center of the University of Maryland

[2] Computer Science Center, University of Maryland, College Park, MD 20742

[3] Department of Mathematics and Statistics, University of Pittsburgh, Pittsburgh, Pittsburgh, PA 15260

NFEARS
A Nonlinear Adaptive Finite Element Solver [1]

Part II: User's Manual
(Version 6)

by

Charles K. Mesztenyi [2] and Werner C. Rheinboldt [3]

[1] This work was in part supported by the Office of Naval Research under contracts N-00014-80-C-9455 and N-00014-85-K-0169 and by the National Science Foundation under grant DCR-8309926. Acknowledgment is also made for the partial support of the Computer Science Center of the University of Maryland

[2] Computer Science Center, University of Maryland, College Park, MD 20742

[3] Department of Mathematics and Statistics, University of Pittsburgh, Pittsburgh, PA 15260

Preface to the User's Manual v. 6

This represents the second Part of the report on NFEARS, the "Nonlinear Finite Element Adaptive Research Solver" developed jointly by the Universities of Maryland and Pittsburgh. This part constitutes the User's Manual for the system version 6. It was intended to describe all necessary aspects for running NFEARS successfully without requiring a detailed knowledge of the mathematical background given in Part I. However, the reader should be generally familiar with the aims and tasks of the program.

Major changes from previous versions are as follows:

1. Optional creation of "Moving Output Files".
   The program allows the user to create binary or ASCII output files consisting of user defined data at each solution point during the single parameter continuation or two parameter region calculation. In case of the region calculation, it is implemented in the REGION subcommand mode, and the output file is called as "Moving Frame Output File". For single parameter continuation calculation, the "POLYG"-on command has been implemented placing the program in polygon mode. The available subcommands in the polygon mode are a subset of the main commands. The output file created during the polygon mode is called "Moving Polygon Output File".

2. Error Calculation.
   One of the main purpose of NFEARS is to investigate the reliability/efficiency of various error calculations. For this, the new version allows more than one error calculations (presently two) but only one of them is used when a new solution point is obtained. With the CONST command, the user can decide which is to be used. Errors are always calculated for newly obtained solution points for the purpose of recording it in the moving output files, thus the new version does not have separate ERROR command. Reference to the type 2 error calculation:

   W.C.Rheinboldt and J. Liu:
   A Posteriori Error Estimates for Parametrized Nonlinear Equations
   Dept. of Mathematics and Statistics
   University of Pittsburgh
   March 16, 1990

   Reliability/efficiency of the finite element error calculation can be measured when the analytic solution of the problem is known. Version 6 of NFEARS requires a user supplied routine to calculate the analytic solution and its derivatives whenever it is known (otherwise a dummy subroutine). The sign of the problem number (IDPR) indicates whether analytic solution is known (negative) or not (positive).

CONTENTS

1. NFEARS Program

1.a Platforms

NFEARS version 6. is available for VAX (VMS) and Unisys Computers.In order, to use the program the user is required to write subroutines in Fortran 77 (i) describing the problem to be solved (see Sections I.1 and II.3), (ii) specifying necessary outputs and to combine them with the NFEARS program. Although, for the most part, NFEARS is written in standard Fortran 77, some special features are assumed to be available in the compiler. The principal non-standard feature is the use of the INCLUDE statement which allows for tne inclusion of program-segment-files.

1.b Interactive/Batch mode of operation

NFEARS has been designed for interactive use such that the user is prompted to give commands for the types of operation to be performed. For a given command, the program also prompts the user for additional input if necessary. To accomodate batch processing, the program provides an option such that the user may collect all interactive inputs (together with the prompts) in an ASCII file which can be edited for a future batch processing. For example, the user may run NFEARS on a VAX performing a few iteration steps with less stringent accuracy requirement; then the optionally generated input file can be edited where the accuracy requirement is specified, the number of iterations steps can be increased, and finally the edited input file can be submitted for a Cray version of NFEARS as a batch input file.

1.c Size of problems to be solved

NFEARS uses labeled common-storage areas extensively for its internal data structure. All of these labeled common-storage areas are defined by declarations in individual files which are then inserted into the program files by means of INCLUDE statements. One of these individual files, MAXDIM, declares parameter values which in turn are used for dimensioning various arrays. These parameter values limit the current size of the problem. If a given problem exceeds these limitations, MAXDIM should be edited for larger values, and NFEARS should be recompiled. The following limitations are presently set up in MAXDIM:

M0MAX	= 16	Maximum number of 0-D domains
M1MAX	= 12	Maximum number of 1-D domains
M2MAX	= 5	Maximum number of 2-D domains
M1TMAX	= 200	Maximum number of free 1-D nodes
M2TMAX	= 225	Maximum number of free nodes in one 2-D domain
M21MAX	= 300	Maximum number of free nodes in one 2-D domain and its boundary
M01DFX	= 8000	Maximum size of the Jacobian corresponding to the free nodes in the 0/1-D domains
M2DFMX	= 44000	Maximum size of the Jacobian corresponding to the free nodes in one 2-D domain and its boundary

1.d Files used

Other possible machine or installation dependent parts of the NFEARS program occur in the program segment file IOPROG in connection with the handling of disk files. When NFEARS is run the following disk files are used by unit numbers :

5	fs	System input file
6	fs	System output file
9	us	Used to equate user's named files
10	fs	Log-file for interactive runs
11	fs	File to copy user's inputs during interactive run which can be used for input file in batch run
12	ud	2-D tree file
13	ud	2-D vector file
14	us	Neumann condition assembly file
15	us	Element assembly file
16	ud	2-D Jacobian file
19	s	Moving data output file ("u" or "f" by user's choice)
20	us	Temporary node data file
21-29	us	Region center-point save files

(u=unformatted, f=formatted, s=sequential, d=direct access)

Units 9, 10, 11 and 19 are associated with user supplied file names. All other units (except 5 and 6) are temporary files. Unit 9 is used to save internal data (see SAVE command) which can be used to restart NFEARS (see RESET command). It is also used to equate geometry input data file. Unit 10 is used for recording a session with NFEARS in interactive processing mode. This allows to print large amount of data without placing them on the terminal screen. It is left to the user to print this file out after the session. Unit 11 is used either to echo user's input in an interactive processing mode, or as an input data file in batch processing mode. Beside actual data, this file can also contain comment lines (asterisk in column 1). In fact, all prompts for user input are recorded when this file is generated in an interactive processing mode. Unit 19 is used to record consecutive iteration steps suitable for postprocessing.

Before any use of NFEARS the user is required to perform the following two steps:

(a) To prepare the geometry input describing the domain Ω, and

(b) to write the user supplied subroutines describing the mathematical problem, specifying the data to be written on moving output files, routine to calculate analytic solution, and to combine them with NFEARS in the form of an executable module.

These steps are described in detail in Sections II.2and II.3

2. Geometry Input Preparation

The preparation of the geometry input consists of the following six steps:

(a) Subdivision of the domain Ω.
(b) Assignment of directions for the 1-D domains.
(c) Numbering of all subdomains.
(d) Definition of an initial mesh.
(e) Specification of an initial solution.
(f) Construction of the geometry input file.

These steps are illustrated with a simple example in Figures 3.1 -3.3.

(a) Subdivision of the domain Ω:

As discussed in Section I.2, the domain Ω must be subdivided into generalized quadrilaterals each with four corner points and four sides. As before, we call the open quadrilaterals 2-D domains (Ω^2_k, k=1,...,N_2), the open sides 1-D domains (Ω^1_k, k=1,...,N_1), and the corner points 0-D domains (Ω^0_k, k=1,...,N_0). Any 1-D domain is either a side of exactly one 2-D domain in which case it is part of the external boundary of Ω, or it is a side of two 2-D domain in which case it is contained in the interior of Ω. As shown in Figure I.2.1, angles formed at the corner points of the 2-D domains should be between α and $180-\alpha$ degrees with a suitable tolerance α to avoid numerical instabilities; a value of $\alpha\approx 15^{o}$ has been found adequate.

Figure 3.1 shows an example where the domain is a quarter disk with Dirichlet boundary conditions on the horizontal line (fixed boundary) and Neumann conditions on the rest of the boundary. The right side of the figure shows a possible subdivision into three 2-D domains, nine 1-D domains and seven 0-D domains. Thus, in this case, we have N_2=3, N_1=9 and N_0=7. It should be noted that, in line with our definition of the admissible meshes in Section I.3, a basic mesh Δ is automatically introduced on Ω once the initial subdivision is given; namely, the mesh consisting exactly of 4 superelements on each 2-D domain.

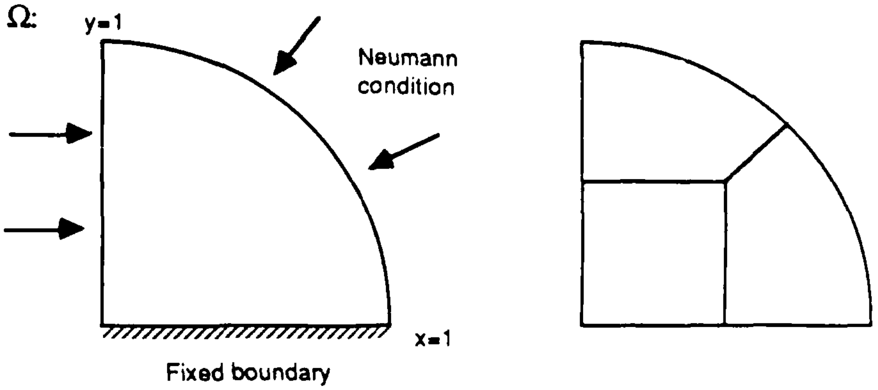


Figure 3.1

(b) Assignment of directions for the 1-D domains.

As detailed in Section I.2, directions have to be assigned to all 1-D domains in order to define their tangent and normal vectors and also their curvature C. On 1-D domains which carry Neumann conditions this assignment must be uniform in the sense that all normals point either outward or inward to the domain Ω and, hence, are not mixed. As discussed in Section I.2, the normal vector is obtained from the tangent vector by rotating the latter counter-clockwise through 90°. Figure II.3.2 shows a possible assignmement of directions for our example.

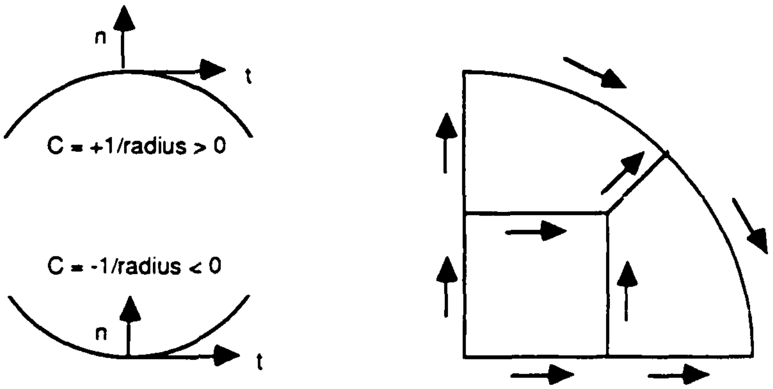


Figure II.3.2

(c) Numbering of all subdomains.

The next step is to number all 0-D, 1-D and 2-D domains. Within each group the numbers should start with 1, and end with N_0, N_1 and N_2, respectively. The order of the 0-D and 2-D domains is irrevelant. However, some savings in speed and memory space can be achieved if the 1-D domains are numbered as follows: Begin by numbering the 1-D domains which carry Dirichlet conditions, then continue with the others by using a "wavefront" to move over the subdivision. Figure II.3.3 shows such a numbering for our example.

(d) Definition of an initial mesh:

In NFEARS meshes are specified in terms of density functions and intensity values. Section I.4 presents the definition of the density function D_Ω on the domain Ω and gives an algorithm for the construction of the mesh from D_Ω and the given intensity $\mathfrak{J}$. The un-normalized density d_Ω is specified in terms of 29 coefficients $p_0,\ldots,p_9$ for each closed 2-D domain. In order to simplify the definition of the starting mesh, NFEARS reduces this input requirement by asking only for a few of these coefficients and by performing linear interpolation to get all others. More specifically, NFEARS requires one coefficient value for each 2-D domain, one for each 1-D domain, and two for each 0-D domain. The single coefficient values for the 2-D and 1-D domains are assigned to the mid-points of these sub-domains as their appropriate p_i-value, $1 \leq i \leq 9$. The first of the two coefficient values for a 0-D domain is again used as their p_i-value, $1 \leq i \leq 9$, while the second coefficient is the p_0 value which describes the singularity. Note that the p_0 values must be either zero or negative. Figure I.3.4 shows the mesh generated from the indicated initial density values for a closed 2-D domain. More specifically, the initial coefficients are shown adjacent to the corners for the 0-D domains (where the second line is p_0), along the sides for the 1-D domains, and near the mid-point of the 2-D domain. With each picture, the intensity is listed. It is advisable to start with uniform coefficient values, and then to increase the $p_1,\ldots,p_9$ values in areas where a singularity is expected.

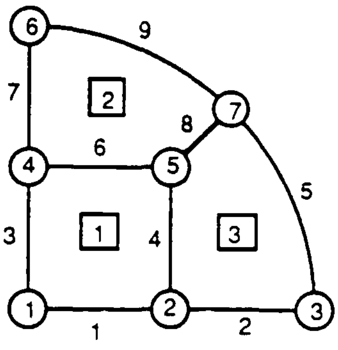
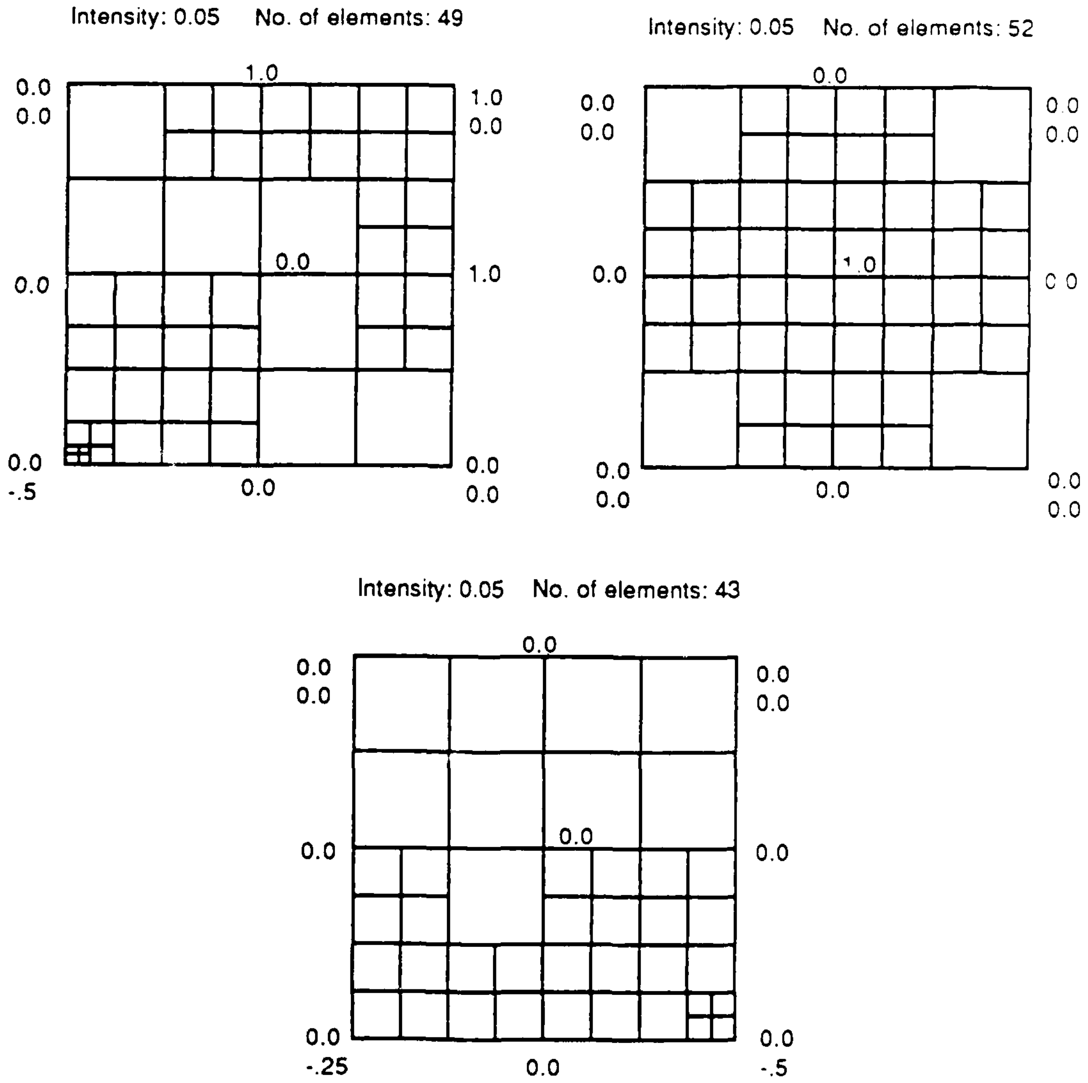


Figure 3.3


Intensity: 0.05 No. of elements: 49

Intensity: 0.05 No. of elements: 52

Intensity: 0.05 No. of elements: 43

Figure 3.4

(e) Specification of an initial solution.

For all calculations NFEARS requires an starting solution on the nodes of the initial mesh (see Section I.6). Once again, in order to simplify the input, NFEARS asks only for a reduced number of solution values and uses biquadratic interpolation to determine the other ones. More specifically, one value is required for each 0-D domain, one each at the mid-points of the 1-D domains, and one each at the mid-points of the 2-D domains. The biquadratic interpolation is based on the local coordinate system as defined in Section I.2. It should be noted that the initial solution values also specify the Dirichlet boundary conditions on the relevant 1-D domains; in other words, they define the corresponding boundary functions b as quadratic functions in the local coordinates (see Section I.2). In our example, we assumed a zero initial solution and zero Dirichlet conditions.

(f) Construction of the geometry input file.

NFEARS permits either an interactive input of the geometry or a read-in of a prepared geometry input file. In order to avoid typing errors it is generally advisable to set up a geometry input file. This input file has to consist of $N_0+N_1+N_2+3$ data lines in free format where, again, N_0, N_1 and N_2 denote the number of 0-D, 1-D and 2-D domains, respectively. The general format is as follows:

N_0	N_0 = Number of 0-D domains;
$1,x_1,y_1,b_1,u_1,p_1,p_1^0$	index i of the i-th 0-D domain;
$2,x_2,y_2,b_2,u_2,p_2,p_2^0$	x_i,y_i = global coordinates of the 0-D domain;
....	b_i = 0 if this 0-D domain is free,
....	=1 if it carries a σ_4-dependent Dirichlet
....	condition ,
....	=2 if it carries a fixed Dirichlet condition ;
....	u_i = initial solution value;
....	p_i = density coefficient;
....	p_i^0 = singularity coefficient
$N_0,x_{N0},y_{N0},b_{N0},u_{N0},p_{N0},p_{N0}^0$	
N_1	N_1 = Number of 1-D domains;
$1,J_1,K_1,b_1,C_1,u_1,p_1$	index i of the i-th 1-D domain;
$2,J_2,K_2,b_2,C_2,u_2,p_2$	J_i, K_i = indices of the adjacent 0-D domains
....	(from - to),
....	b_i = 0 this 1-D domain is free,
....	= 1 if it carries a σ_4-dependent Dirichlet
....	condition,
....	= 2 if it carries a fixed Dirichlet condition,
....	= -1 if it carries a Neumann condition;
....	C_i = signed curvature of the 1-D domain;
....	u_i = solution value at its mid-point
....	p_i = coefficient of the density function at the
....	mid-point;
$N_1,J_{N1},K_{N1},b_{N1},C_{N1},u_{N1},p_{N1}$	
N_2	N_2= Number of 2-D domains;
$1,I_1,J_1,K_1,L_1,u_1,p_1$	index i of the i-th 2-D domain;
$2,I_2,J_2,K_2,L_2,u_2,p_2$	I_i,J_i,K_i,L_i = indices of the adjacent 1-D
	domains
....	ordered counter clockwwise;
....	u_i = initial solution at the mid-point
....	p_i = coefficient of the density function at the
....	mid-point
$N_{N2},I_{N2},J_{N2},K_{N2},L_{N2},u_{N2},p_{N2}$	

Notes:

1. When a 1-D domain carries a Dirichlet boundary condition (b_i= 1 or 2), then the two bounding 0-D domains should have the same type of boundary condition.
2. The definition of the sign of the curvature for a 1-D domain is indicated in Figure II.3.2; that is, if we look from the starting 0-D domain , J_i, toward the terminating 0-D domain, K_i, then the positive (+) sign or negative (-) sign is to be used when the center of the circle is on the right or the left side, respectively. For straight lines, the value of the curvature is zero.
3. The indices of the four 1-D domains that bound a 2-D domain have to be listed in counter-clockwise order. When the 2-D domain is mapped into the units-quare, the first 1-D domain is mapped into the η axis and the second one onto the ξ axis (see Section I.2).

For our example, the input file has the following form:

```
7                              Number of 0-D domains
1,0.,0.,2,0.,0.,-.5            Data for the seven 0-D domain
2,.5,0.,2,0.,0.,0.
3,1.,0.,2,0.,0.,0.
4,0.,.5,0,0.,0.,0.
5,.5,.5,0,0.,0.,0.
6,0.,1.,0,0.,0.,0.
7,.70710678,.70710678,0,0.,0.,0.
9                              Number of 1-D domains
1,1,2,2,0.,0.,0.               Data for the nine 1-D domains
2,2,3,2,0.,0.,0.
3,1,4,-1,0.,0.,0.
4,2,5,0,0.,0.,0.
5,7,3,-1,1.,0.,0.
6,4,5,0,0.,0.,0.
7,4,6,-1,0.,0.,0.
8,5,7,0,0.,0.,0.
9,6,7,-1,1.,0.,0.
3                              Number of 2-D domains
1,3,1,4,6,0.,0.                    Data for the three 2-D domains
2,7,6,8,9,0.,0.
3,4,2,5,8,0.,0.
```

3. User Supplied Subroutines

Before running NFEARS, the user must write subroutines which

a. calculate the values and derivatives of the functions Φ, G_2, G_1 in the problem-definition of Section I.1, and which set up or modify certain parameters and print out appropriate headings.

b. supply those values which to be placed into the Moving Data Output files.

c. calculate the analytic solution/derivatives when it is known, otherwise a dummy subroutine.

All these subroutines carry entry names beginning with USR... .Note also that all of them must be provided even if some are not in use, since the some operating systems do not handle missing subroutines. Once these routines have been written and compiled, they have to be combined with NFEARS to produce an executable module.

NFEARS provides a common block

/USRPAR/ FUSER(10,2), IUSER(10)

for up to 20 real and 10 integer valued parameters for use in the supplied subroutines. The subroutine USRFCT allows for storing of data in these two arrays during the initial call while USRMOD permits their later modification. These data are retained and may be used in all other user subroutines. They are also saved by a SAVE command , and read back by a RESET command. Other internally used common blocks may be used by these routines but the contents of those are not saved. In this case, it is recommended labels USRn with numeric digits "n" to avoid naming conflict with other common blocks internal with NFEARS.

The following subroutines must be provided:

Problem definition routines

USRFCT This routine is called at initialization time. It may also be used to print out some captions.

USRINV Routine to provide an initial solution for the problem. It is called by the "INVAL" command.

USRMOD Routine to provide, change, or print user parameter in USRPAR during the process.

USRPH1 Routine to provide the first derivatives of the function Φ at specified points.

USRPH2 Routine to provide the second derivatives, including the derivatives by σ_1, of the function Φ at specified points.

USRG1 Routine to provide the values and derivatives by σ_3 of the function G_1 at specified points.

USRG2 Routine to provide the values and derivatives by σ_2 of the function G_2 at specified points.

Moving Data File routines:

USRPY0 Routine to provide the number of data to be written out for each iteration step during single parameter continuation (POLYG mode). It is called when the file is to be established.

USRPYS Routine to perform any initialization needed before one iteration step, e.g. clearing summation fields.

USRPYE Routine to perform any calculation needed when an element with its nine point solution values are provided.

USRPYF Routine to perform any calculation needed when all elements have been processed for this iteration step.

USRFR0 Same as USRPY0, but used for two-parameter interations (REGION mode).

USRFRS Same as USRPYS but in REGION mode.

USRFRE Same as USRPYE but in REGION mode.

USRFRF Same as USRPYF but in REGION mode.

Analytic solution routine:

USTRUX Routine to provide analytic solution if known, otherwise a dummy routine.

When NFEARS is used in interactive mode such that it generates an input file to be edited for future batch processing, it is unable to copy user's input initiated from these user supplied subroutines. In this case, it outputs a "comment" warning line on the generated file indicating that possible input line should be inserted during the editing session. This is before the following subroutines are called from NFEARS:

USRFCT when it is called frim INIT the first time
USRINV, USRMOD, USRPY0, USRFR0 whenever they are called

3.a Problem defining routines

The calling sequences are as follows:

SUBROUTINE USRFCT (I, NU, IDPR, IDUF, IDUFP, IDUG2, IDUG1)

This subroutine is called at the initialization of a problem either by an INIT command or a RESET command. It is expected to print out a caption for the run.

Input arguments:

I = 0 for initial start with INIT,
= 1 for recall of saved data with RESET

NU = Fortran unit number (6 or 10) where echo print should be directed

Output arguments (if I=0):

IDPR = Problem number, should be an integer: negative if analytic solution is known, positive if it is not known

IDUF = ID number of the Φ function

IDUFP = 0 if Φ does not depend on σ_1, non-zero otherwise

IDUG2 = signed identification number of G_2 as follows:
= 0 if zero; that is, if the G_2 term does not exist
< 0 if G_2 is independent of σ_2
> 0 if G_2 depends on σ_2

IDUG1 = signed identification number of G_1 as follows:
= 0 if zero; that is, if the G_1 term does not exist
< 0 if G_1 is independent of σ_3
> 0 if G_1 depends on σ_3

As noted before, in all cases, the program should print a title for the run. The other arguments should be set by the user if I=0. They will have been set before when I=1, but may be reset by the routine. NFEARS merely checks whether they are zero, positive or negative integers.

SUBROUTINE USRINV (N, XG, U)
DIMENSION XG(2,N), U(N)

This subroutine provides initial solution values U at N points in one 2-D domain with the coordinates x,y specified in the array XG. This routine is called by the INVAL command. The routine is called first with N=0 to allow for any initialization, such as the input of a file of data. Thereafter it is called for all regular free nodes of the problem. Note that an initial solution can be specified by the geometry input in which case USRINV may be a dummy subroutine. But then the INVAL command should never be used.

SUBROUTINE USRMOD (NU)

This subroutine allows for the modification and print-out of the parameter values in the common block /USRPAR/. It is called by the command UMOD. The input integer NU is the Fortran unit number (6 or 10) where the printed output should be directed.

SUBROUTINE USRPH1 (N, IX2, S1, XG, U, P)
DIMENSION S1(2), XG(2,N), U(0:2,N), P(0:2,N)

This subroutine evaluates the first derivatives of Φ at N points in one 2-D domain, and returns the results in the array P.

Input arguments:

N = number of points where the first derivatives of Φ are to be evaluated

IX2 = index value of the 2-D domain containing the points

S1 = the components of the parameter σ_1

XG(1,K), XG(2,K)
= global coordinates x,y of the point K (K = 1,...,N)

U(0,K) = the solution value u at the point K

U(1,K) = the value of the derivative $u_x = \partial u/\partial x$ at the point K

U(2,K) = the value of the derivative $u_y = \partial u/\partial y$ at the point K

Output arguments:

P(0,K) = $\partial\Phi/\partial u$ at the point K (K=1,...,N)

P(1,K) = $\partial\Phi/\partial(u_x)$ at the point K (K=1,...,N)

P(2,K) = $\partial\Phi/\partial(u_y)$ at the point K (K=1,...,N)

SUBROUTINE USRPH2 (N, IX2, S1, XG, U, PU, PL)
DIMENSION S1(2), XG(2,N), U(0:2,N), PU(0:2,0:2,N), PL(2,0:2,N)

This subroutine evaluates the second derivatives of Φ at N points in one 2-D domain, and returns the results in the arrays PU and PL.

Input arguments:

N = number of points where the second derivatives of Φ are to be evaluated

IX2 = index value of the 2-D domain containing the points

S1 = the components of the parameter σ_1

XG(1,K), XG(2,K)
= global coordinates x,y of the point K (K = 1,...,N)

U(0,K) = the solution value u at the point K

U(1,K) = the value of the derivative $u_x = \partial u/\partial x$ at the point K

U(2,K) = the value of the derivative $u_y = \partial u/\partial y$ at the point K

Output arguments:

PU(I,J,K) = $\partial^2\Phi/\ \partial U(I,K)\ \partial U(J,K)$, (I,J=0,1,2; K=1,...N)
PL(I,J,K) = $\partial^2\Phi/\ \partial S1(I)\ \partial U(J,K)$, (I=1,2; J=0,1,2; K=1,...,N)

SUBROUTINE USRG1 (N, IX1, S3, XG, CN, G, GL)
DIMENSION S3(2), XG(2,N), CN(2,N), G(N), GL(2,N)

This subroutine calculates the function G_1 defining the Neumann boundary conditions, and its derivatives by σ_3, at N points in one 1-D domain, and returns the results in the arrays G and GL.

Input arguments:

N = number of points where the evaluation is to take place
IX1 = index value of the 1-D domain containing the points
S3 = the components of the parameter σ_3
XG(1,K),XG(2,K)
= global coordinates x,y of the point K (K = 1,...,N)
CN(1,K),CN(2,K)
= components of the normal unit vector at K in the global coordinate system.

Output arguments:

G(K) = the value of G_1 at the point K
GL(J,K) = the derivatives $\partial G(K)/\partial S3(J)$, J=1,2 of G_1 by σ_3 at the point K.

SUBROUTINE USRG2 (N, IX2, S2, XG, G, GL)
DIMENSION S2(2), XG(2,N), G(N), GL(2,N)

This subroutine evaluates the value of the function G_2 and its derivatives by σ_2 at N points in one 2-D domain, and returns the results in the arrays G and GL.

Input arguments:

N = number of points where the evaluation is to take place

IX2 = index value of the 2-D domain containing the points

S2 = the components of the parameter σ_2

XG(1,K),XG(2,K)

= global coordinates x,y of the point K (K = 1,...,N)

Output arguments:

G(K) = the value of G_2 at the point K

GL(J,K) = the derivatives $\partial G(K)/\partial S2(J)$, J=1,2 of G_2 by σ_2 at the point K.

3.b Moving Data File routines

NFEARS provides the generation of sequential files (formatted or unformatted) to record data associated with solutions obtained during single parameter continuation (POLYG mode) and two parameter frame calculation (REGION mode). The following user supplied subroutines should supply the actual data to be recorded. Up to 50 real valued data are allowed for which NFEARS provides a single array DATNOD for the user's subroutines. While the calling sequences of the two types of (POLYG and REGION) are similar, the format of the generated files are different which are described in Appendix C. It is assumed that these files serve for inputs of various postprocessors. There are no separate user supplied subroutines subroutines for closing these files. NFEARS automatically close these when the user exits from the POLYG and/or REGION subcommand mode.

The number of arguments in the calling sequences of the following routines are kept to a minimum. Since the variety of data values to be recorded can be quite extensive, Appendix A lists the available subroutines (GET...) which can be called from these subroutines to obtain various data otherwise internal to NFEARS. Appendix B provides some examples for construction of these routines.

SUBROUTINE USRPY0(NDLNGT)

This routine is called initially when the user request a polygonal data output file to be generated.

Output argument:
NDLNGT = Number of real valued data to be recorded

SUBROUTINE USRPYS(NDLNGT,DATNOD)
DIMENSION DATNOD(50)

This routine is called after a solution data has been obtained but no data to be recorded is processed. It serves as an initialization process for the next two routines. The most common use is to initialize some entries in DATNOD which are to be used summation and/or minimum/maximum calculationby the next routine.

Input argument:
NDLNGT = Number of real valued data to be recorded (received from USRPY0)

Output argument:
DATNOD = array provided to record data values

SUBROUTINE USRPYE(NDLNGT,DATNOD,ID2,IDE,H,XL,U)
DIMENSION DATNOD(50), XL(2), U(3,3)

This routine is called for each finite element at the obtained solution node.

Input argument:
NDLNGT = Number of real valued data to be recorded (received from USRPY0)
ID2 = Index of the 2-D subdomain where the element is
IDE = Index value of the element (unique in the ID2 subdomain)
H = Side-length of the element (in the mapped unitsquare)
XL = Local coordinates (in the unitsquare) of the middle point of the element

U = solution values at the nine points of the element

Output argument:
DATNOD = array provided to record data values

SUBROUTINE USRPYF(NDLNGT,DATNOD)
DIMENSION DATNOD(50)

This routine is called after USRPYE is called for each element at the obtained solution point. It is intended to be used to finalize data collected with USRPYE, also to store data values in DATNOD which are globally available for this solution point, e.g. error estiimates for the full domain at this solution.

Input argument:
NDLNGT = Number of real valued data to be recorded (received from USRPY0)

Output argument:
DATNOD = array provided to record data values

SUBROUTINE USRFR0(NDLNGT)

Same as USRPY0

SUBROUTINE USRFRS(NDLNGT,DATNOD)
DIMENSION DATNOD(50)

Same as USRPY0

SUBROUTINE USRFRE(NDLNGT,DATNOD,U,K)
DIMENSION DATNOD(50), U(3,3)

Same as USRPY0

SUBROUTINE USRFRF(NDLNGT,DATNOD)
DIMENSION DATNOD(50)

Same as USRPY0

3.c Analytic solution routine

```
SUBROUTINE   USTRUX(X,SPAR,CPARM,U,DX)
DIMENSION   X(2),CPARM(2,4),DU(2)
```

This routine should supply the analytic solution and its derivatives at the point specified by the X argument when it is known, otherwise it can be a dummy subroutine.

Input arguments:
X(1),X(2) = x,y coordinates where the solution, derivatives are needed
SPAR = value of the single λ parameter
CPARM = values of the σ parameters

Output argument:
U = solution value
DU(1) = $\partial U/\partial x$
DU(2) = $\partial U/\partial y$

4. Running NFEARS

NFEARS can be used either interactively or in batch mode. The first integer valued input from system input file 5 defines the mode of operation as follows:

-1 The program is to run in batch mode, the next input from unit 5 must be the name of the filefrom which subsequent are to be taken. This file will be equated to unit 11.

0 The program is to run in interactive mode, the next input must be a file-name to be used as Log-file where detailed output can be written. This file will be equated to unit 10, and it is left to the user to print it out after the session with NFEARS. Unit 11 is not used.

+1 The program is to run in interactive mode as above, but in addition all inputs with their prompts will be written out on unit 11. This mode requires requires the user to give two file names, first for the Log-file (unit 10), second for the Input-echo file (unit 11).

Once the mode of operation is established, the program enters into a main command mode as illustrated in Figure 4.1. The command names can be typed in lower or upper case. Once a command is given, the program may prompt for further input, specific to that command. After a command has been successfully executed, NFEARS prints out the execution-time and then prompts for a new command input. These prompts also indicate whether the program is in main, polygon or region command mode. The user may insert comment lines after an input prompt before giving the actual data by having an asterisk (*) in column 1 followed by the comment string. NFEARS writes out these comment lines on the appropriate output files: unit 6 for batch mode, unit 10 for interactive mode, unit 11 if Input -echo file is to be generated.

In interactive mode, the Log-file provides a record of the NFEARS session and may contain a large amount of data which could not be handled conveniently on the terminal's screen. This file will be created by NFEARS as a new, sequential, formatted file. Thereafter, output strings are written onto it, and an end-of-file termination occurs when the QUIT command is given. It is the user's responsibility to print or discard this file after termination of the NFEARS run. The prompt for a command-input, the time spent for the execution of a command, and any potential error messages will always show up at the terminal. The amount of output can be set and modified by the "TRACE" command.

Two of the available commands, POLYG and REGION, places NFEARS into a subcommand mode which are described separately. In this Section we summarize briefly the essential aspects of these commands and refer to Section 5 and 6 for more detailed descriptions of each of them.

CONST to set certain constants for the continuation algorithm and for the mesh modifications, and to re-define the mesh intensity $\mathfrak{I}$,

INIT to initialize a problem,

INVAL to change existing values of the initial solution,

HELP to print out all command names as well as the last command,

TRACE to specify the amount of output,

RESET to reset NFEARS from a previously saved file,

STEP to step along the solution path with the continuation algorithm,

MESH to modify the mesh,

TARG to request a target and/or limit point calculation,

UMOD to call the user supplied subroutine USRMOD,

PARAM to modify certain parameters,

PRINT to print out solutions, errors, meshes, etc,

SAVE to save present data on a file,

POLYG to record individual steps in the use of continuation algorithm,

REGION to calculate an approximation of some region of the solution manifold,

QUIT to terminate the current run with the program.

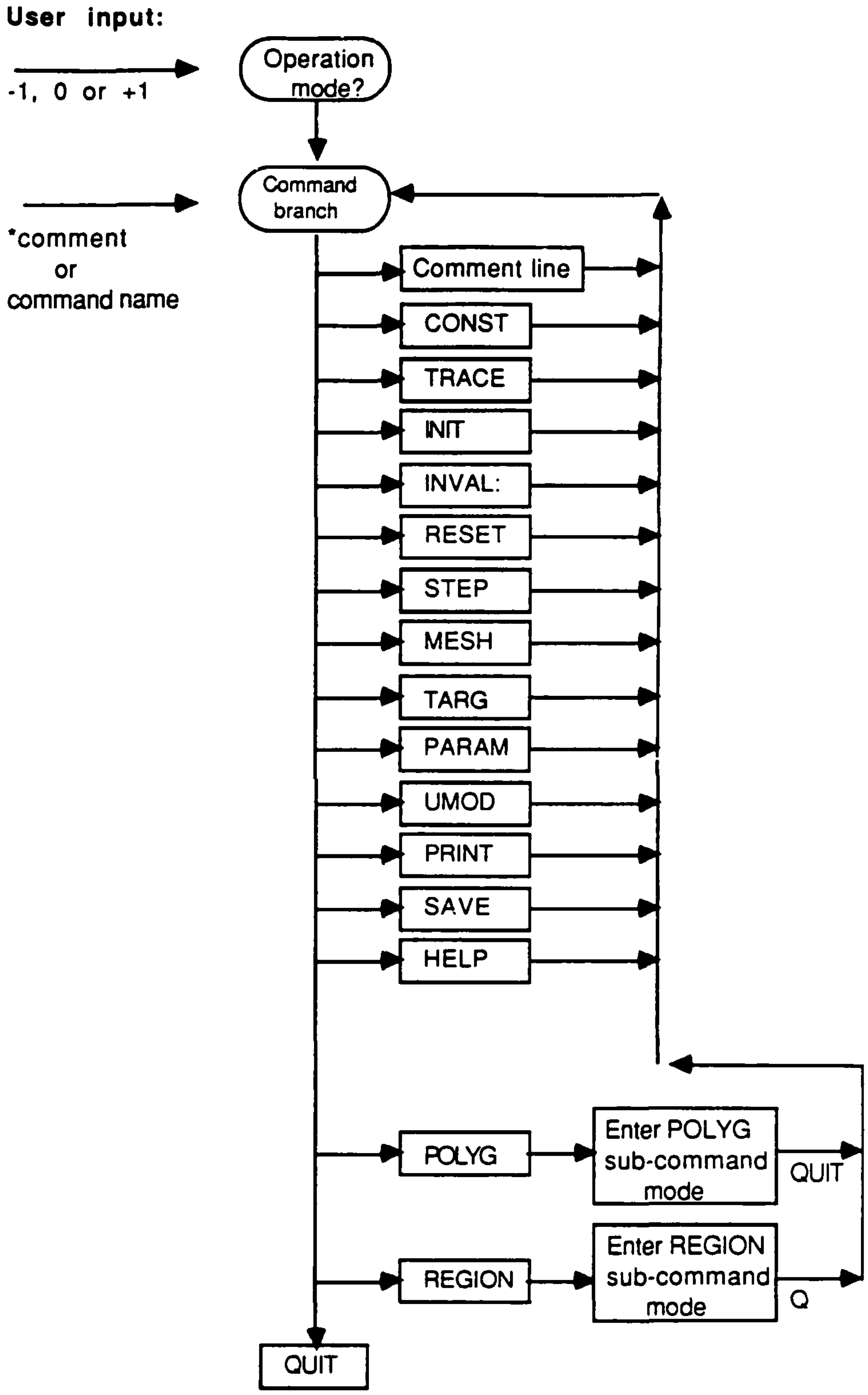

User input:
-1, 0 or +1
Operation mode?
*comment
or
command name
Command branch
Comment line
CONST
TRACE
INIT
INVAL:
RESET
STEP
MESH
TARG
PARAM
UMOD
PRINT
SAVE
HELP
POLYG
Enter POLYG sub-command mode
QUIT
REGION
Enter REGION sub-command mode
Q
QUIT

Figure II.4.1

Order of Commands:

Although NFEARS will, in general, execute commands as they are given, there are certain logical order which should be observed in the sequence of the commands. First of all, one has to establish the problem data either by the INIT or RESET commands. When the problem data (Geometry, etc.) is to be established by the INIT command, it should be preceded by CONST command and followed by a STEP command specifying zero as number of steps, i.e.

CONST -----> INIT -----> (INVAL) ---->STEP -----> ...

CONST assures that necessary constants for correcting the initial values are correctly set, and the STEP command with zero number of steps performs the correction. The INVAL command may be inserted if initial values are supplied by user supplied subroutine.

In case of starting with the RESET command, i.e. getting the problem data from a previously saved file, the CONST command should be given after the RESET command,

RESET -----> CONST -----> ...

since the saved file contains these data values, but it is always advisable to include this command since it prints out the values of these constants before asking for any changes and hence provides a printed record of them in the log-file.

The second observation to be followed is that the REGION command should be preceded an followed by the PARAM command:

... -----> PARAM -----> REGION -----> PARAM -----> ...

This order assures that the effective parameters, λ_1 and λ_2, are reset to zero.

5. NFEARS Commands

In this section we discuss the individual NFEARS commands in detail.

5.1 The CONST Command:

The CONST command permits a change of certain control parameters for the continuation algorithm (STEP commands), the region calculation (REGION command), the mesh modification (MESH command) and error calculation. When this command is invoked, the user is asked to opt either for the constants of group 1 used in STEP, REGION, or for those of group 2 needed in MESH and error calculation., and allowed to change their values. Initially NFEARS sets these to some default values. The various constants, and, in parentheses, their default values are as follows:

(a) Group 1: Constants for the CORR, STEP and REGION commands:

Maximum number of steps allowed per STEP call (5)
Starting step size (0.01)
Maximum step size (1.0)
Minimum step size (0.0001)
Maximum number of steps in the corrector iteration (10)
Frequency of Jacobian evaluation (3)
Absolute error tolerance for the corrector iteration (10*C)
Relative error tolerance for corrector iteration (10*C)
Minimum pivot value allowed in matrix decomposition (C)

where C is the smallest number on the computer such that 1.0+C is different from 1.0.

(b) Group 2: Constants for the MESH command:

Choice of error calculation mode*
Mesh-modification mode: "manual" or "automatic" (automatic)
Control of the "automatic" mesh-modification: "by error size" or "by density" (by density)
Tolerances for automatic mesh-modification by "error size":

Refinement tolerance: If the error indicator of an element exceeds this tolerance then the element is subdivided. De-refinement tolerance: If ω is an element of a previous mesh for which all four sons are elements of the current mesh and their combined error indicators fall below this tolerances, then ω is de-refined.

Intensity, initially set by user's input.

For both groups of constants, the program prints out the presently set values, and then asks if any of them should be changed.

* Presently, there are two modes of error calculation. The first (default) mode is the one used in the previous versions of NFEARS and described, the second mode is based on calculating the error indicators of elements by subdividing them and solving the linearized system over the element with zero boundary condition such that the solution is ortogonal to the tangent(s). This mode also gives error term(s) for the parameter value(s). Mathematical reference is given in ...

5.2 The TRACE Command:

This command sets the Indicator for the amount of output from NFEARS and may be invoked at any time. It prompts for two, free-formatted input lines:

(i) The first line consists of one integer valued data:

ISTCOM

where

ISTCOM ≥ 0 indicator in interactive mode for the amount of print-out to be directed to the terminal (system output file 6). A zero value keeps it to a minimum, and for increasing positive values the amount of output is increased. In batch mode, this has no effect since all output is directed to unit 6.

(ii) The second line consists of six non-negative integers corresponding to 6 specific parts of NFEARS:

I1 , I2 , I3 , I4 , I5 , I6

These 6 parts are identified as follows:

I1: Execution of the INIT and RESET commands

I2: Corrector iteration

I3: Execution of the MESH command

I4: Execution of the CORR and STEP commands

I5: Execution of the FERR command

I6: Execution of the REGION Command

The integer value I_k ($1 \leq k \leq 6$) specifies the amount of output from Part k that is to be generated in the log-file on unit 10, and, when ISTCOM $\geq I_k$, also on the system output file 6, whenever NFEARS is running in interactive mode. In batch mode, all specified output is directed to the system output file 6.

5.3 The INIT Command:

This command initializes a new problem. NFEARS first calls the user-subroutine USRFCT with the first argument set to zero. This call allows for any set-up of parameter values that may needed in other user-subroutine and also for the print-out of a problem title.

Upon return from USRFCT, the program asks for the name of the file containing the geometry data. If the answer is "5", then these geometry data are to be given interactively. Otherwise, the answer is assumed to be the file-name where the geometry data reside with the format descibed in Section II.2.

After the geometry input, the program asks for the intensity $\mathfrak{I}$ of the initial mesh. This value can be changed again by the CONST command.

Thereafter, the program asks for the coefficients of the effective parameters. First, the coefficients δ_1 and δ_2 with $\delta_1+\delta_2 = 1.0$ of the linear functions $\lambda_1 = \delta_1\lambda$, $\lambda_2 = \delta_2\lambda$ are requested which specify the continuation-path (see Section I.5). Then the eight coefficients α and β are expected that define λ_1 and λ_2 in terms of the problem parameters (see Section I.1).

In summary, the input for the INIT command is as follows:

1. Input from USRFCT;
2. name of the geometry file or "5" for interactive input;
3. geometry data if the input is to be interactive;
4. intensity $\Im$
5. δ_1 , δ_2
6. k, α_k^1 , β_k^1 , k=1,2,3,4

   k, α_k^2 , β_k^2 , k=1,2,3

After the INIT command, the user should apply either a STEP or an INVAL---STEP sequence of commands to correct the initial values. For the STEP command, the user should specify zero as number of steps. (The STEP command with zero steps replaces the CORR command used in previous versions of NFEARS)

5.4 The INVAL Command:

This command is used to supply an initial solution through the USRINV subroutine. When this command is invoked, USRINV is called first with the input variable N set to zero to allow for any initialization that may be needed in this subroutine. Thereafter, USRINV is called repeatedly with N>0 to obtain initial values U at N nodes with global coordinates (x,y). Note that this command cannot be used before the geometry is established by the INIT or RESET commands. Moreover, the INVAL command should always be followed by the STEP command with zero number of steps to correct the supplied solution and to establish it and its related data in the temporary storage area.

5.5 The RESET Command:

The RESET command causes the data structure from a previously saved disk-file, generated by a SAVE command, to be read back into the program. The user must supply the name of the file. After the file is read, NFEARS calls USRFCT with the first argument set to one to allow for a print-out of a problem title and, if desired, of any saved parameters in the common block /USRPAR/.

This command should be followed by a CORR command to establish the solution and its related data in the temporary data structure and to ensure its correctness. Once again, the INVAL command may be applied prior to CORR.

5.6 The STEP Command:

The STEP command invokes the continuation algorithm and prompts the user for the number of steps that are to be taken. If zero number of steps is specified, then NFEARS attempts to correct the solution values given either at the initial data input during INIT command or received by the INVAL command. Otherwise, this number can be given as a positive or negative integer. If given as a positive integer then an error calculation is performed after each step automatically; if given as a negative integer then error calculation is performed only after the last step. It terminates in either one of the following three modes:

(a) The specified number of steps have been taken. The "current" location contains the "point" on the path reached at the last step.

(b) During the step-calculation, a previously called-for target or limit point is detected between two successively computed points on the path. The solution corresponding to this target or limit point is returned in the "predictor" location of the temporary data structure while the computed point on the path just beyond it is in the "current" location.

(c) The corrector iteration failed to converge.

A print-out informs the user which of these modes applies, and, in particular, whether a target or limit point has been found. It is advisable not to take too many steps with one STEP command, since it may happen that the discretization errors become undesirably large. When the corrector iteration fails, the user may try to decrease the minimum step size of the continuation algorithm by means of the CONST command. Another remedy might be to establish a more suitable scaling of all variables and, especially, of the parameters λ_1 and λ_2; but, of course, this requires some modifications in the user-subroutines.

5.9 The MESH Command:

This command checks the present meshes on the 2-D domains and modifies them in accordance with the mesh-modification-mode set by the CONST command. The program tests first if any de-refinement is needed; that is, whether any four elements obtained by a prior refinement of an element are to be contracted again into one element. Once all de-refinements, if any, are completed, the program checks if any refinement is to be performed; that is, whether any element is to be subdivided into four elements.

Modification is performed in accordance with the mode set by the CONST command. "Automatic" refinement decision can be made on the basis of the error-sizes or of the density and intensity. The refinement by error-size uses two error tolerances supplied by the CONST command and for details of the refinement by density/intensity we refer to Section I.8. "Manual" modification is performed interactively. The user is asked for each element, which is a candidate for de-refinement or refinement, whether the particular operation should be performed or not. In all cases, the total number of de-refinements and refinements is provided as output.

Interpolation is used to obtain the solution values for any nodes that may have been newly established by any refinement. The resulting approximate solution should always be corrected by a CORR or STEP command.

5.8 The TARG Command:

This command prints out any presently set target and/or limit point indicators, if there are any, and then asks for new indicators if these are to be established. A target or limit point indicator always consists of the index value of the variable or parameter variable, and, in addition -- for a target point -- of the desired target value. The following variable indices are allowed:

A 0-D domain not carrying a Dirichlet boundary condition,
the mid-point of a 1-D domain not carrying a Dirichlet boundary condition,
the mid-point of a 2-D -domain,
any one of the active λ or σ parameter variables.

5.9 The PARAM Command:

This command prints out the present values of the effective parameter values λ_1, λ_2 and of the coefficients δ_1, δ_2, α and β. It then asks if the δ and β values are to be changed. The following answers are allowed:

0 , 0 (two zeros) no change,
δ_1 , δ_2 ($\delta_1+\delta_2 = 1.0$) change the previous δ_1 and δ_2 values,

1 , β reset the value of β_i^k for the printed i,k indices.

In either case, new values of the α_i^k are determined which ensure that the current values of the program parameters σ_i^k are not changed, and then the values of the effective parameters λ_1,λ_2 are set to zero. More specifically -- after all changes are provided -- the parameter values will be as follows

$$\alpha_i^k \text{ (new)} = \alpha_i^k \text{ (old)} + \beta_i^k \text{ (old) } \lambda_k \text{ (old)} \quad , \quad i=1,\dots,4;\ k=1,2$$

$$\lambda \text{ (new)} = \lambda_k\text{(new)} = 0, \quad \beta_i^k \text{ (new)} = \text{input provided by the user.}$$

In addition, any previously set target or limit point indicators are erased. Thus , if desired, such target and limit point indicators have to be reset in terms of the new λ-values initialized to 0.

5.10 The UMOD Command:

This command invokes a call to the user-subroutine USRMOD. Accordingly, any action taken depends on this routine.

5.11 The PRINT Command:

This command prints out the current solution in the tree-storage area and its associated error and density values. It should be used after a FERR command, otherwise the program may print a previous result. When this command is given, the program asks whether output should be on-line (system output 6). If the answer is "No", then the full solution, error and density values will be printed in the log-file (unit 10). If the answer is "Yes" (on-line), then the program will prompt further for a specification of the desired segments of the solution, error and density data which are to be printed out for each subdomain.

5.12 The SAVE Command:

This command saves the present data on a disk-file which can be used later to resume the computation by means of a RESET command. The program asks for the name of the file (which should not exist) to which the output is to be directed. The program saves all problem data including the parameter values in the user-subroutines. The data in the temporary storage area; that is, in particular, the Jacobian and the path-tangents, are not saved. Accordingly, this command should be given after a FERR or MESH command. It is the user's responsibility to save this disk file permanently after the termination of an NFEARS session.

5.13 The HELP Command:

This command prints out the names of the commands and the name of the last executed command.

5.14 The POLYG Command:

This command places NFEARS into a sub-command (polygon) mode. These sub-commands are a subset of the main commands as illustrated in Figure 5.1 The only differences are that the "STEP" as subcommand performs only one step (thus no further input is asked for the number of steps to be performed), the "HELP" command lists only the available subset of commands, and the "QUIT" sub-command returns NFEARS to the main command flow. When NFEARS prompts for input of a command, it distinguishes between the main command mode and polygon subcommand mode by printing "MAIN COMMAND" or "POLY COMMAND". The main purpose of this mode is to record results of of each step in a moving data output file. When this command is invoked, NFEARS asks if moving polygon data file is to be generated which requires input character "Y" or "N" (yes or no), and if it is yes, then specifying a file name for the moving data file.

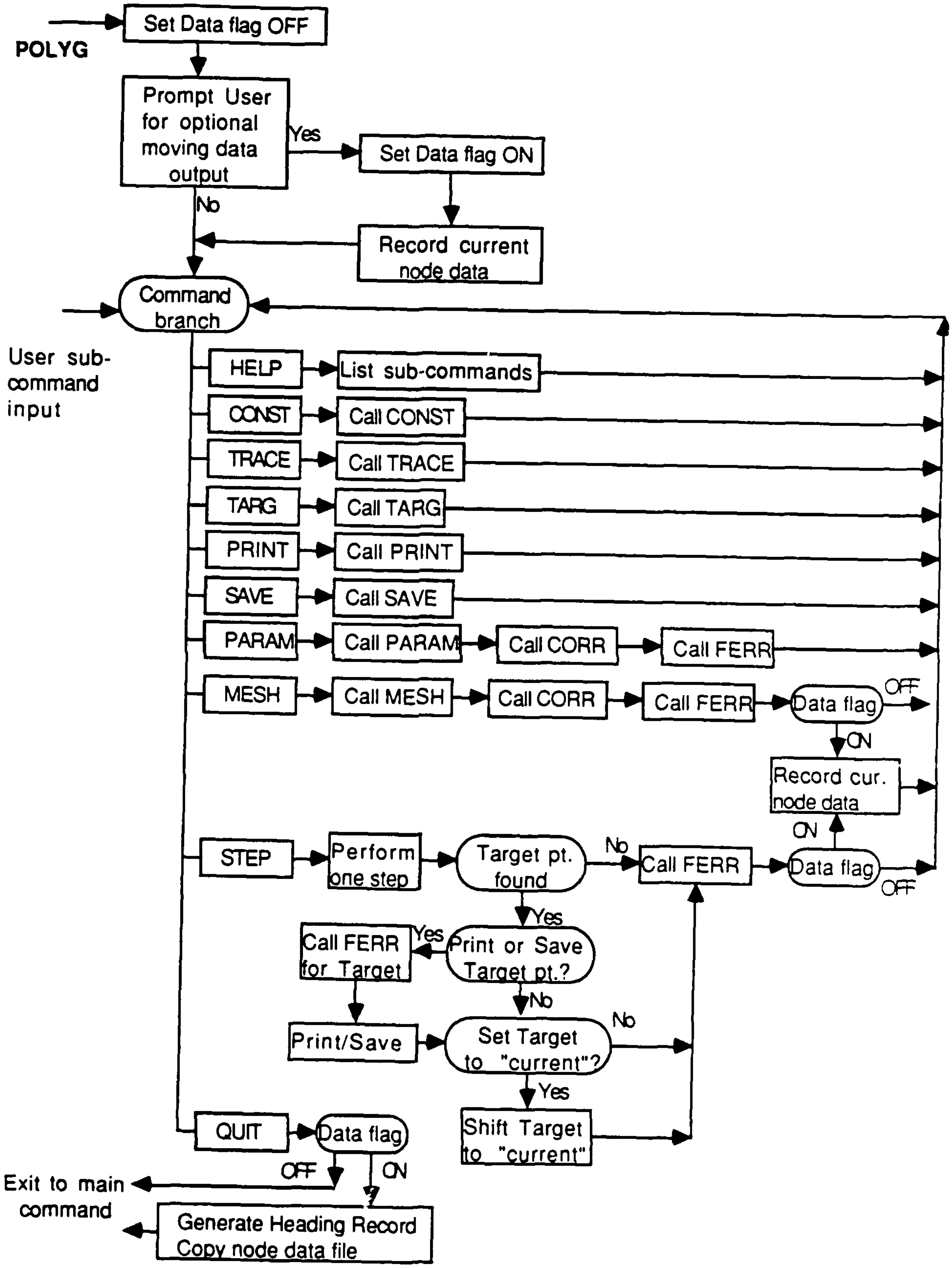

POLYG
Set Data flag OFF
Prompt User for optional moving data output
Yes
Set Data flag ON
No
Record current node data
Command branch
User sub-command input
HELP
List sub-commands
CONST
Call CONST
TRACE
Call TRACE
TARG
Call TARG
PRINT
Call PRINT
SAVE
Call SAVE
PARAM
Call PARAM
Call CORR
Call FERR
MESH
Call MESH
Call CORR
Call FERR
Data flag
OFF
ON
Record cur. node data
ON
STEP
Perform one step
Target pt. found
No
Call FERR
Data flag
OFF
Yes
Call FERR for Target
Yes
Print or Save Target pt.?
No
Print/Save
Set Target to "current"?
No
Yes
Shift Target to "current"
QUIT
Data flag
OFF
ON
Exit to main command
Generate Heading Record Copy node data file

Figure 5.1

5.17 The REGION Command:

This command invokes the algorithm for the calculation of a simplicial approximation of an open region of the manifold, and enters into a sub-command mode which is described in the next section. Before invoking this command, the user should ensure that both effective parameter variables, λ_1 and λ_2 are active. This can be guaranteed with the PARAM command by providing that at least one β_i^1 and at least one β_i^2 are non-zero. Accordingly, upon exit from the REGION sub-command mode, the user may wish to invoke the PARAM command to choose a new relation between the two parameter variables.

When this command is invoked, the user is asked whether a moving frame data file should be generated requiring users input "Y" or "N". If user's input is Y then the program also asks for a name of the file.

5.18 The QUIT Command:

This command terminates the NFEARS session and prints out the number of region-output files (see Section II.6) that have been generated. These files and any files generated by a SAVE command should be saved by the user, and , in addition, the log-file (unit 10) should be printed out.

6. REGION Subcommands

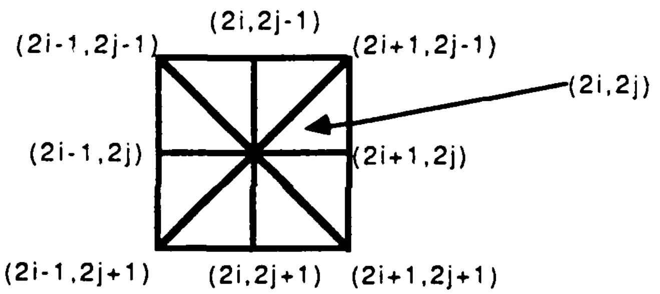
The REGION command calculates a simplicial approximation on the manifold M in an neighborhood of the "current solution" called, in this context, the reference point. If a presently available target point is to be used as the reference point, then it has to be placed into the "current" location by issuing a FERR and CORR command prior to the REGION command. The Region sub-program is implemented as a routine which is controlled by user supplied sub-commands each of which consists of a single character. When the REGION command is invoked, (i) the program asks the user whether a moving frame data file should be generated. If the answer is "Y" (yes) then it also asks for a name of this file; (ii) the program checks whether both effective parameters are active, if not then a request for a change of the parameter dependence is issued. When this condition is satisfied, the program resets the values of the two effective parameters λ_1 and λ_2 to zero.

The basic principle of the region calculation is discussed in Section I.7. As noted there a Kuhn triangulation in R^2 is used as the reference triangulation and we work with rectangular patches of eight triangles each containing one center node and eight boundary nodes. As outlined in Section I.7, the algorithm begins by mapping a first patch onto the tangent plane of the reference point and by projecting the nodes from there onto the manifold. Then, under user control, the program proceeds to transfer an adjacent patch, next to the first one, onto the manifold, etc. In other words, the REGION command maps a sequence of "patches" onto the manifold, and their connectivity pattern defines the desired simplicial approximation of the particular region.

The moving frame data file from the REGION command consists of two heading files followed by a sequence of records containing the attributes of the calculated nodal points as calculated by the user supplied subroutines, USRFRx, as described in Section 3.c. The second heading file contains the connectivity matrix NODE(i,j), i,j=1,...,MAXNOD (=21), The entries of the connectivity matrix correspond to the nodes of a 20 by 20 subset of triangles of the reference triangulation (see Figure I.7.1); the triangular connections are not stored explicitly. Originally, this matrix is set to zero, except for the entry NODE(10,10) which corresponds to the node that is mapped into the reference point and is set to 1 which serves also as its index number. When a node of any patch has been mapped successfully onto M and its attributes have been calculated, then the node receives a positive index which is recorded in the matrix. At the same time, the attribute record of that point is saved. A negative index is used in the connectivity matrix when an attempt has been made to transfer the point to M but the corrector iteration has failed. In the matrix, the center-points of the patches correspond to the "even" entries, NODE(2i,2j) $(1\leq i,j\leq 10)$, and the boundary points of the patch are the eight adjacent matrix entries. Thus a patch is identified by nine entries NODE(2i+m,2j+n), m,n=-1,0,+1. The adjacent patches, of course, share the boundary nodes.

Indices of a patch in the NODE matrix

Center point in NODE(2i,2j)



After the initialization, the program enters into the sub-command mode. The order in which patches are to be transfered onto M is controlled by the sub-commands L(eft), R(ight),D(own) and U(p), i.e. the patch to be used next is the one adjacent to the "current" patch in the specified direction. Figure II.6.2 shows the sequence of patches corresponding to the commands L,D,R,R,R. The "current" patch is usually the last calculated one. But, the program also provides nine temporary locations, indexed 1 to 9, where the center points on M of a successfully mapped patch can be saved by means of the sub-command S(ave). The reference point is initially saved in location 1. During the calculation another sub-command allows for a previously "saved patch" to replace the "current" one. When the region computation terminates, control is returned to the main program with a "current" solution. At that time, the user also has the choice which of these saved center points should become the "current" solution.

Generation of patches:

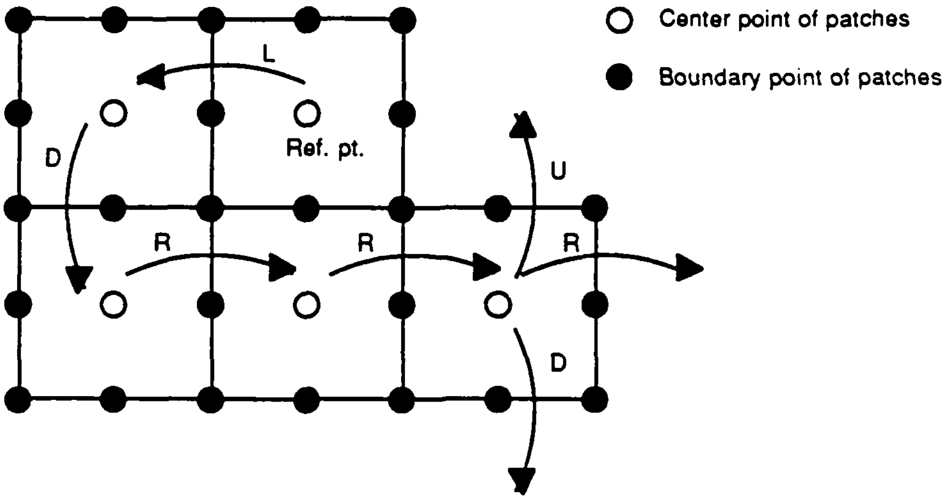


Figure II.6.2

The computation of the center-point needed for mapping a new patch onto M and the process of transfering a node from R^2 onto the tangent plane and of projecting it from there onto M was discussed in Section I.7. Default step sizes are based on the last stepsize of the continuation algorithm and can be changed by the user. For the center- point calculation the modified Newton process is the same as in the continuation algorithm, but the program always re-evaluates the Jacobian after the solution has been obtained. The calculation of the boundary points of the new patch uses a chord Newton method based on the Jacobian at the center point of the patch. A flow chart of the region-subprogram is given in Figure II.6.3.

A character matrix picture of the calculated patches is printed at the terminal. The entries in this matrix correspond to the center points of the patches. and consist of a character followed either by a blank or a single digit between 1 and 9. Patches which were not yet mapped onto M are represented by a period followed by blank. Instead of the full 10 by 10 array of patches, only the used patches are shown with one row and/or column of periods on the four sides where applicable. The characters representing these patches are as follows:

or "R" = Initial reference patch

or "A" = Successfully transfered patch for which all 9 nodes have been mapped onto M.

or "B" = Unsuccessfully used patch for which the corrector diverged at one or more nodes

The "current" patch is indicated by a capital letter and all others by small letters. A digit between 1 and 9 following the patch character indicates that the center-point for that patch has been saved in the temporary location with the same index.

Example:

Initial ouput picture:

```
.    .    .
.    R1   .
.    .    .
```

The current patch is the reference patch; its center point is the reference point which is always saved in location 1. The next patch may be chosen in any one of the four directions (Left, Right, Up, Down). Suppose that at a later time the picture looks as follows:

```
.    b    a    a    A    .
.    a    a    a    .    .
.    a    a    a    .    .
.    a2   a    a    .    .
.    a    r1   a    .    .
.    .    .    .    .    .
```

Only two movements are possible from the current patch "A", namely -- as indicated by the missing period in the top line -- the R(ight) and the D(own) movement . The center points of two patches were saved in locations 1 and 2. All patches were mapped successfully onto M, except the one in the upper left corner.

Below the picture, the program prints out a line of summary data, including the number of patches used so far, number of nodes transfered onto M, etc, and, optionally, three 3 by 3 matrices which give some information about the current patch calculation. For details we refer to the P(rint) sub-command below. These matrices can also be printed automatically with a proper setting of the T(race) sub-command.

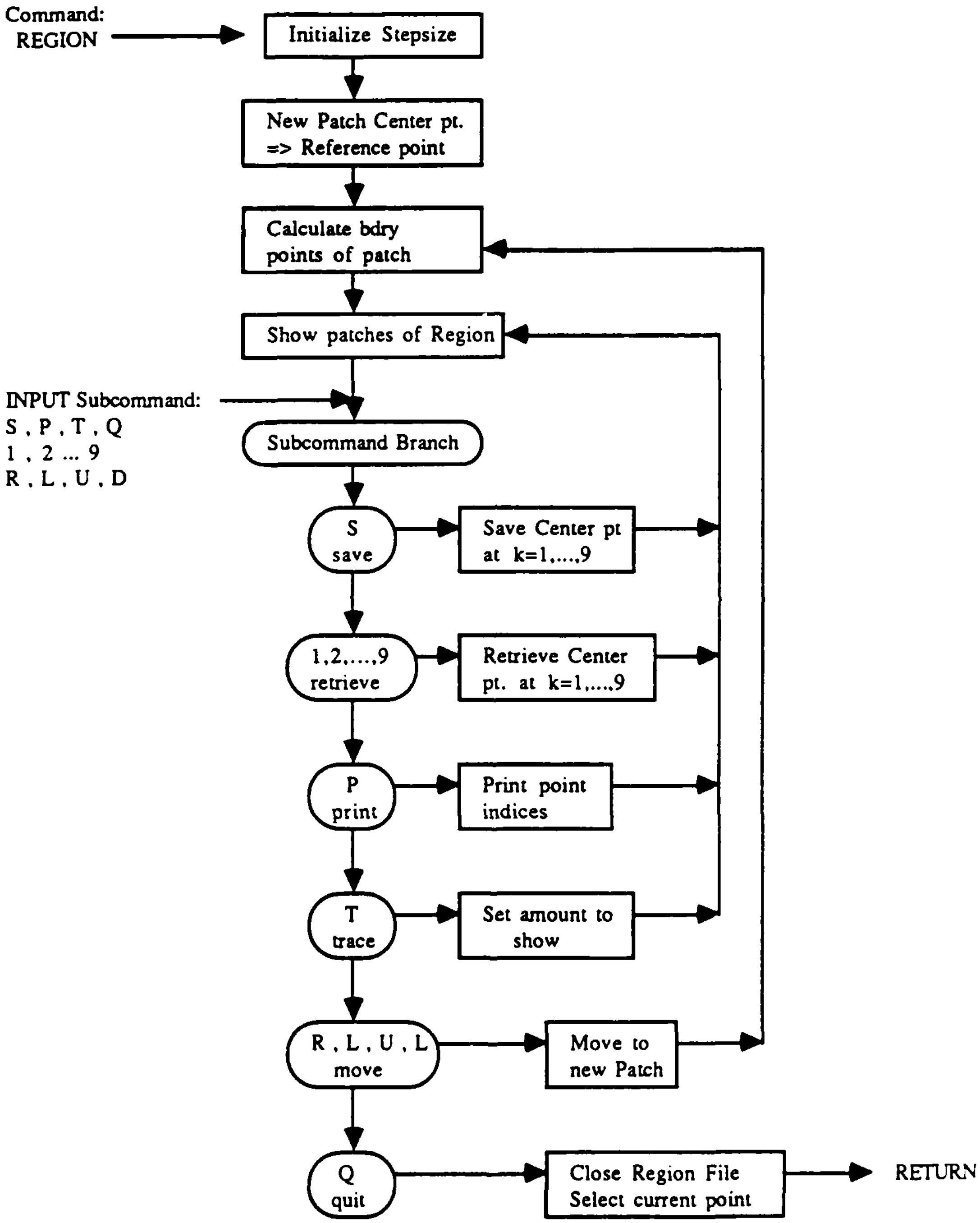


Figure II.6.3

After printing the picture the program expects another one-character (sub)command from the user. Following is a list of all these subcommands:

= Save the current patch: The center-point of the current patch is saved in the temporary location with the next available index. If all 9 locations have been used then the user will be asked which one of them is to be replaced. The index of all patches for which the center points have been saved can be seen in the next picture.

, "2", ..., "9"

= Use the "saved patch" in this location as the "current patch"; an error message results if the indicated location is empty.

= Prints three 3 by 3 matrices are printed side by side which give information about the last patch calculation. The first matrix contains the indices of the nodes that were mapped onto M, the second one shows the number of corrector steps for their calculation , and the third matrix contains the distance, in the maximum norm, from the predicted point on the tangent space to the computed node on M.

When the Print command is given, the program asks whether the above output should be directed to the terminal or to file 10. It may be noted that the T(race) command also allows for the automatic generation of the same output, except that then it will always be directed to the terminal.

, "R", "U" or "D"

= Calculates a new patch adjacent to the current patch by moving in the indicated direction L(eft), R(ight), U(p) or D(own). If the patch in that position has already been mapped onto M, or if it is outside of the 10 by 10 patch region, an error message is issued. Once the new patch has been transfered onto M, it will become the new current patch.

= Trace command provides for certain optional outputs. It requires two integers as inputs, which are the same as the trace switches of the main program for the amount of terminal output. A 1,1 input produces an automatic output of the same three matrices on the terminal as the P(rint) command, while 0,0 suppresses this print-out.

= Quit command: It establishes a region-file before returning control to the main program. The user has a choice which of the saved center-points of the patches is to become the "current" solution. This solution should be "corrected" by a CORR command.

Appendix A

GETxxx Subroutines

A set of subroutines to obtain various data internal to NFEARS is provided by the "moving data file" routines, USRPYx and USRFRx. These routines are grouped into two parts: (1) Routines callable from USRxxS or USRxxF, (2) Routines callable from USRxxE. Actually, the first set of routines could also be called from the USRxxE routines but that usage would be very inefficient. When any of those data are needed for the USRxxE routine, one should call that routine from USRxxS and save the needed data in a locally defined common storage. The names of all subroutines start with GET... , and the data provided by them correspond to the "current" (last obtained) solution.

A.1 Subroutines for USRxxS/USRxxF

These routines can be called when a solution has been obtained before the elements being processed (from USRxxS) or after all elements has been processed (from USRxxF). The USRxxS and USRxxF routines are called with the argument list:

```
(ND,D)
DIMENSION D(ND)
```

where D is the array for storing data to be recorded.

```
DIMENSION   SIGMA(2,4)
........
CALL   GETSST(NP,NE,SIGMA)
```

GETSST returns the following data:

NP = Number of free variables (nodes) in Ω

NE = Number of elements in Ω

SIGMA = σ parameter values

```
DIMENSION  XSP(2),PR(2),AP(2,4),BP(2,4)
.....
CALL  GETSLB(ISP,SP,XSP,PR,AP,BP)
```

GETSLB returns the following data:

ISP = 1 in polygon mode, 2 in region mode
SP = value of the single parameter λ when combined
XSP = coefficients for combining $\lambda 1$ and $\lambda 2$:
$\lambda = XSP(1) * \lambda 1 + XSP(2) * \lambda 2$
PR = values of $\lambda 1$ and $\lambda 2$
AP, BP = coefficients defining σ values:
SIGMA(I,J) = AP(I,J) + BP(I,J) * PR(I)

```
CALL  GETSER(IET,ER,ERTR)
```

GETSER returns global error data:

IET = error calculation type (1 or 2)
ER = total error for Ω
ERTRU = true error for Ω when analytic solution is known

```
CALL  GETSMX(EMAX,EMIN,DMAX,DMIN)
```

GETSMX returns maximum and minimum error informations. The max. indicators show whether element subdivision is needed, the min. indicators show whether element contaction is needed:

EMAX = max. elemental error indicator (squared)
EMIN = min. sum of four elements' squared error indicators
DMAX = intensity for the element with EMAX
DMIN = intensity for the 4 elements with EMIN

```
DIMENSION   SD(0:2,2)
.....
CALL   GETSSD(SD)
```

GETSSD return min/max values of the solution and its derivatives:

SD(0,1)	= min U
SD(0,2)	= miax U
SD(1,1)	= min ∂U/∂x
SD(1,2)	= max ∂U/∂x
SD(2,1)	= min ∂U/∂y
SD(2,2)	= max ∂U/∂y

A.2 Subroutines for USRxxE

USRxxE routines are called with the argument list:

```
(ND,D,ID2,IDE,H,XL,U)
DIMENSION D(ND),XL(2),U(3,3)
```

giving the following informations about the element :

D = array for storing data to be recorded
ID2 = index of the 2-D subdomain where the element is
IDE = index of the element (unique in the 2-D subdomain)
H = side-length of the element in the unit-square
XL = coordinates of the middle point of the element in the unitsquare
U = solution values at the middle-, side- and corner-points of the element

Some of these values are needed in the following GET... subroutines as input arguments,

```
DIMENSION   XG(2),XI(2)
.....
CALL   GETELC(XG,XI)
```

Given the global coordinates of a point in XG which must be in the same 2-D subdomain as the one specified by ID2, GETELC returns the corresponding local coordinates of the point in XI when the 2-D is mapped into the unit-square.

```
CALL  GETEER(EI2,EP1,EP2,ETR)
```

GETEER routine gives the following error informations about the element:

EI2 = error indicator squared (without parameter component)
EP1 = parameter 1 component of the elemental error
EP2 = parameter 2 component of the elemental error
ETR = true elem. error indicator squared if analytic solution is known

EP1 and EP2 is applicable when errors are calculated by type 2.

```
DIMENSION  U(3,3),XG(2,3,3),VN(0:2,3,3),DWH(3,3)
        .....
CALL  GETEQD(U,XG,VN,DWH)
```

GETEQD requires input argument U, solution values at the middle-, side- and corner-points of the element, which is given in the argument list of USRxxE. The routine then gives the following 9-point Gaussian quadrature data where I,J indices correspond to the Gaussian points in the element (I,J=1,2,2)

XG(1,I,J),XG(2,I,J) = global x,y coordinates
VN(0,I,J) = solution values at (I,J)
VN(1,I,J) = $\partial U/\partial x$ (x=global coord.) at (I,J)
VN(2,I,J) = $\partial U/\partial y$ (y=global coord.) at (I,J)
DWH(I,J) = absolute value of the Jacobian determinant of the 2-D mapping multiplied by the appropriate weight and by the square of the side length of the element

This routine can be used when the user needs to integrate a functional f(x,u,u') over the element.

```
DIMENSION  XE(2),U(3,3),XG(2),SD(0:2)
.....
CALL  GETESD(XE,U,XG,SD)
```

GETESD routine returns point-wise information about the finite element solution. It requires input data XE and U where XE contains the coordinates of the point when the element is mapped into unitsquare, and U contains the solution values at the middle-, side- and middle-points of the element. GETESD gives the following data:

XG	= global x,y coordinates of the point
SD(0)	= finite element solution u(x,y) at the point
SD(1)	= ∂u/∂x at the point
SD(2)	= ∂u/∂y at the point

This routine can be used for checking the accuracy of the finite element solution at selected points, possibly close to singularity. Note that XE is different from the 2-D local coordinate, but can be derived from it:

Let XI(1),XI(2) be the local coordinates of a point in a 2-D domain when it is mapped into the unitsquare. Then

XE(1) = 0.5 + (XI(1)-XL(1))/H
XE(2) = 0.5 + (XI(2)-XL(2))/H

where XL, H are the middle point coordinates in 2-D, and side length of the element given in the argument list for USRxxE. The point is inside of the element if

$$0 < XE(i) < 1, \quad i=1,2$$

Appendix B

Example for USRPYx subroutines

The following example is for the user supplied subroutines, USRPYx, illustrating the data collection/calculations for the moving polygonal data file. The same programs can be used for the USRFRx routines by replacing the entry names.

```
C----------------------------------------------------------------
C DATA RECORDED IN MOVING FILES:
C  1. EFFICIENCY INDEX FOR THE FULL DOMAIN = (FE ERROR)/(TRUE ERROR)
C  2. MIN. ELEMENTAL EFFICIENCY INDEX
C  3. MAX. ELEMENTAL EFFICIENCY INDEX
C  OPTIONAL OUTPUTS IN GROUPS OF SIX DATA (MAX. 5 GROUPS)
C  4,5,6 = FE SOLUTION U, dU/dx, dU/dy  AT POINT 1.
C  7,8,9 = TRUE SOLUTION T, dT/dx, dT/dy AT POINT 1.
C  MAXIMUM 5 POINTS ALLOWED
C
C STORAGE NEEDED IN COMMON AREA /USRDAT/:
C   DIGMA(2,4) = PRESERVING SIGMA VALUES
C   EFMNX(2) = RETAINING MIN/MAX ELEMENTAL EFFICIENCY
C   MPRT = NUMBER OF POINTS WHERE SOLUTIONS AND DERIVATIVES
C          CALCULATED (MAX. 5)
C   I2P(2,5) = 2-D INDICES AND ELEMENT INDICES OF THE ABOVE POINT
C   XGI(2,5) = GLOBAL COORDINATES OF THE ABOVE POINTS
C   XLI(2,5) = 2-D LOCAL COORDINATES OF THE ABOVE POINTS
C
C**************************************
C S.8 LENGTH OF POLYGONAL NODE RECORD *
C**************************************
      SUBROUTINE USRPY0(LENGTH)
      COMMON/USRDAT/MPRT,I2P(2,5),DIGMA(2,4),EFMNX(2),XGI(2,5),XLI(2,5)
C INPUT FOR POINTS WHERE SOLUTION/DERIVATIVES NEEDED
10    WRITE (6,20)
20    FORMAT (' INPUT number of points where sol/der. to be recorded',
     : ' (max. 5) ==>')
      READ (5,*,ERR=10) MPRT
      IF ((MPRT.LT.0).OR.(MPRT.GT.5)) GO TO 10
      IF (MPRT.NE.0) THEN
30    WRITE (6,40) MPRT
40    FORMAT (' INPUT points line-by line: 2-D index, X,Y coord. ==>')
      DO 50 I=1,MPRT
      READ (5,*,ERR=30) I2P(1,I),XGI(1,I),XGI(2,I)
50    CONTINUE
      WRITE (10,60) MPRT,(I2P(1,I),XGI(1,I),XGI(2,I),I=1,MPRT)
60    FORMAT(1X,I2,' points where sol/der. are recorded:'/(I5,2G14.6))
      END IF
C RETURN LENGTH OF NODE RECORD
      LENGTH = 3 + 6*MPRT
      RETURN
      END
C=======================================================
C
C***********************************************
C S.9 INITIALIZE FOR ONE POLYGONAL NODE RECORD *
C***********************************************
C
      SUBROUTINE USRPYS(LENGTH,DAT)
      DIMENSION DAT(LENGTH)
      COMMON/USRDAT/MPRT,I2P(2,5),DIGMA(2,4),EFMNX(2),XGI(2,5),XLI(2,5)
C GET SIGMA VALUES AND SAVE THEM IN DIGMA
      CALL GETSST(NP,NE,DIGMA)
C CLEAR ELEMENT INDICES IN I2P(2, )
      IF (MPRT.EQ.0) THEN
        DO 10 I=1,MPRT
10      I2P(2,I) = 0
      END IF
C INITIALIZE MIN/MAX ELEMENTAL EFFICIENCY IN EFMNX
      EFMNX(1) = 1.E10
      EFMNX(2) = -1.E10
C RECORD DOMAIN EFFICIENCY INDEX
      CALL GETSER(IET,ER,ERTRU)
      DAT(1) = ER/ERTRU
      RETURN
      END
C======================================================
C
C*****************************************
C S.10 UPDATE POLYGONAL DATA BY ELEMENT *
C*****************************************
C
      SUBROUTINE USRPYE(LENGTH,DAT,ID2,IDE,H,XL,U)
      DIMENSION DAT(LENGTH),XL(2),U(3,3)
      COMMON/USRDAT/MPRT,I2P(2,5),DIGMA(2,4),EFMNX(2),XGI(2,5),XLI(2,5)
C INFORMATION AVAILABLE:
C   ID2 = INDEX OF THE 2-D SUBDOMAIN
C   IDE = INDEX OF THE ELEMENT IN THE 2-D
C   H   = SIDELENGTH OF THE ELEMENT
C   XL = LOCAL COORDINATES OF THE MIDDLE POINT OF ELEMENT
C   U = SOLUTION VALUES AT THE CORNER, SIDE AND MIDDLE POINTS
      DIMENSION Z(2),ZG(2)
C
C CALCULATE AND UPDATE ELEMENTAL EFFICIENCY IN EFMNX
      CALL GETEER(ER,ER1,ER2,ERTRU)
      ER = SQRT(ABS(ER/ERTRU))
      EFMNX(1) = AMIN1(EFMNX(1),ER)
      EFMNX(2) = AMAX1(EFMNX(2),ER)
C LOOP ON MPRT POINTS
      IF (MPRT.EQ.0) RETURN
      DO 100 I=1,MPRT
C   SKIP IF DIFFERENT 2-D OR ELEMENT INDEX IS NOT ZERO
C        (IF ELEMENT INDEX WAS SET, THEN WE ALREADY HAVE IT)
      IF ((IABS(I2P(1,I)).NE.ID2).OR.(I2P(2,I).NE.0)) GO TO 100
C   IF 2-D INDEX IS POSITIVE THEN
C   CALCULATE LOCAL COORDINATES AND SET 2-D INDEX TO NEGATIVE
      IF (I2P(1,I).GT.0) THEN
        CALL GETELC(XGI(1,I),XLI(1,I))
        I2P(1,I) = - I2P(1,I)
      END IF
C   IS POINT IN THIS ELEMENT?
```

```
      H2 = H/2.
      IF ((XLI(1,I).GT.(XL(1)-H2)).AND.(XLI(1,I).LE.(XL(1)+H2))
     : .AND.(XLI(1,2).GT.(XL(2)-H2)).AND.(XLI(1,2).LE.(XL(2)+H2)))
     : THEN
C    CALCULATE DATA NEEDED TO BE RECORDED AND SET THE ELEMENT INDEX
      Z(1) = .5 + (XLI(1,I)-XL(1))/H
      Z(2) = .5 + (XLI(2,I)-XL(2))/H
      CALL GETESD(Z,U,ZG,DAT(6*I-2))
      CALL USTRUX(XLI(1,I),Z,DIGMA,DAT(6*I+1),DAT(6*I+2))
      I2P(2,I) = IDE
      END IF
C END OF LOOP
100   CONTINUE
      RETURN
      END
C=========================================================
C
C**************************
C S.11 FINALIZE NODE DATA *
C**************************
C
      SUBROUTINE USRPYF(LENGTH,DAT)
      DIMENSION DAT(LENGTH)
      COMMON/USRDAT/MPRT,I2P(2,5),DIGMA(2,4),EFMNX(2),XGI(2,5),XLI(2,5)
C RECORD MIN/MAX ELEMENTAL EFFICIENCY
      DAT(2) = EFMNX(1)
      DAT(3) = EFMNX(2)
      RETURN
      END
C========================================================
```

Appendix C

Formats of Moving Data Files

The final Moving Data Output file is generated on the user's specified file when the polygon or region subcommand mode is terminated by the by the QUIT or Q command, respectively. The sequential output file is either formatted or unformatted, as the user specified. Formatted file is prefered whenever the file is to be transfered to other platform for postprocessing.

Both types (polygcn and frame) files contain two heading (logical) records followed by the node data records. While the formats of the first heading record and the node records are the same for both polygon and frame files, the second heading record differ. The node data records contain the actual data supplied by the user supplied subroutines.

Header 1.

The record consisting the following 9 data:

NFEARS = (integer) version number of NFEARS
IDPR = (integer) problem number, if negative then analytic solution is available
IDUF = (integer) id. number of Φ
IDUG2 = (integer) id. number of G2
IDUG1 = (integer) id. number of G1
NDTOT = (integer) number of node records
NDLNGT = (integer) length of node records
MAXNOD = (integer) for length used in Header 2.
NDTYPE = (integer) code for types of node records:
= 1 for moving polygon node data record,
= 2 for moving frame node data record,

Formatted record is written out with FORMAT (9I8).

Header 2:

for polygon data file:

The record containing

RNGBUF(i,j), i=1,2 and j=1,NDLNGT

where

RNGBUF(1,j) = min DATNOD(j)

RNGBUF(2,j) = max DATNOD(j)

with min and max is taken over all node data (solution points).

Formatted record is written out with FORMAT (5G14.6)

for frame data file:

Three physical records containing

RNGBUF(i,j), i=1,2 and j=1,NDLNGT

NPTOT,(NDFRST(i),i=1,2),(NDLAST(i),i=1,2)

((NODE(i,j),i=1,MAXNOD),j=1,MAXNOD)

These data are written out as one logical record for unformatted file, and as three logical records for formatted file using the FORMATs

(5G14.6)	for RNGBUF
(9I8)	for NPTOT,NDFRST and
(9I8)	for NODE

RNGBUF contains the same data as for polygon data file. NPTOT is the number of patches visited, the matrix NODE is a connectivity matrix, NDFRST(1) and NDLAST(1) are the first and last non-zero rows in the matrix NODE, NDFRST(2) and NDLAST(2) are the first non-zero columns in the matrix NODE. See further details regarding patches and connectivity in section 6.

Node data records

Node data records contain the data which were generated by the user supplied subroutines

DATNOD(i), i=1,NDLNGT

Formatted records are written with FORMAT (5G14.6). One record for each solution node. The logical index numbers of the solution nodes correspond to the order of the records on the file, starting with 1 and ending with NDTOT.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 18, 1991	3. REPORT TYPE AND DATES COVERED Technical Report

4. TITLE AND SUBTITLE

NFEARS
A Nonlinear Adaptive Finite Element Solver
Part II: User's Manual (Version 6)

5. FUNDING NUMBERS

ONR-N-00014-90-J-1025

6. AUTHOR(S)

Werner C. Rheinboldt
Charles K. Mesztenyi

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Dept. of Mathematics & Statistics, Univ. of Pittsburgh
Computer Science Center Univ. of Maryland

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

ONR

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release : distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This represents the second Part of the report on NFEARS, Version 6, the "Nonlinear Finite Element Adaptive Research Solver" defveloped jointly by the Universities of Maryland and Pittsburgh. This part constitutes the User's Manual for the system version 6. It was intended to describe all necessary aspects for running NFEARS successfully without requiring a detailed knowledge of the mathematical background given in Part I. However, the reader should be generally familiar with the aims and tasks of the program.

14. SUBJECT TERMS

Finite Element Computations
Adaptive Mesh Refinement

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT unclassified	20. LIMITATION OF ABSTRACT

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to ***stay within the lines*** to meet ***optical scanning requirements***.

Block 1. Agency Use Only *(Leave blank)*.

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	PR	- Project
G	- Grant	TA	- Task
PE	- Program Element	WU	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency Report Number. *(If known)*

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of...; To be published in.... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD - See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE - See authorities.
NASA - See Handbook NHB 2200.2.
NTIS - Leave blank.

Block 12b. Distribution Code.

DOD - Leave blank.
DOE - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA - Leave blank.
NTIS - Leave blank.

Block 13. Abstract. Include a brief *(Maximum 200 words)* factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code *(NTIS only)*.

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.